 尽管到 Perl 6 中才宣称完全支持 Unicode，但是在 Perl 5.8.6 中内嵌的对 Unicode 的支持已经足以开始编写本地化的应用程序了。Ted 使用自己所钟爱的编辑器 Yudit，介绍了 Perl 如何读取、解释和处理 UTF-8 编码的 Unicode。 从 5.6.0 版本发布以来，Perl 已经发生了极大的变化。Perl 6 项目是在一个名为 Parrot 的解释器（它内嵌了对 UTF-8 和很多其他与 Unicode 有关的特性的支持）的基础上构建和运行的（我知道这种说法太过简单，但是本文后面马上要介绍的内容可以确保您并不需要担心这些细节）。因此，Perl 6 从一开始就非常重视对 Unicode 的支持。（所以，在 Perl 6 发布时我还会撰写另外一篇有关 Perl 6 的 Unicode 文章。我当然希望这很快就可以实现。到时候我就不会只是这么简单地提一下就算完了！） 在撰写本文时，我参考的是 5.8.6 版本，但是目前最新版本是 5.8.7。我并没有对 CPAN 的任何 Unicode 模块进行测试 尽管这些模块的确存在，而且也可以很好地工作 因为我们在本文中重点讨论的是 Perl 中内嵌的对 Unicode 的支持特性。因此，我所测试的目标是 Perl 的 Unicode（UTF-8 编码）的读取、解释和处理能力。在本文中我也并没有讨论 Unicode、UCS 和 UTF-8 编码的内容 在 参考资料 中可以找到介绍这个主题的丰富资源。 我喜欢使用 Yudit 编辑器和 rxvt（Unicode 版本）的终端来进行有关 Unicode 的工作。（我一直在使用 rxvt，而不是 xterm；它们都可以很好地处理一些简单的事情，但是从我的观点来看

，rxvt 要更好些，因为它可以使用多种字体来显示不同的字符集。）Unicode 的准备 为了这篇使用 UTF-8 编码的文章，我做好了所有的工作。您应该可以在大部分现代 UNIX&reg. 系统中安装 rxvt-unicode 终端包。您还需要确保支持一个 UTF-8 locale，这与 OS 有关。通常来说，您应该可以这样做：setenv LANG en_US.UTF-8. rxvt-unicode，然后就可以了；但是有些系统，例如 Debian Linux，可能需要其他一些步骤来启用 UTF-8 locale（通常它们默认是没有启用的）。另外，en_US.UTF-8 可能对于您来说并不是一个合适的 locale。您可以检查自己的 OS 手册 和 HOWTO 文档。 您应该安装 Yudit 编辑器，它的功能非常强大，而且非常容易使用。您也可以使用 CVS 版本的 Emacs，我所使用的是 2005 年 1 月的版本（当前的稳定版本是 21.4）。Emacs 可以读写 UTF-8 编码的文件，在我的实验中都非常稳定 只是对于高级的 Unicode 工作来说，功能没有 Yudit 强大。注意：在为本文提供的脚本中，已经 包括了 use strict. 和 use warnings. 附注（pragmas）（您可以从下面的 下载 部分下载 cp35.zip 文件，其中包含有这些附注）。一直启用这些附注就不会出现一些导致遗憾的错误。打印 Unicode 字符 关于 Unicode，最基本的事情（也是最有用的事情）可能就是它让程序可以打印超出 ASCII 和 Latin-1 字符集范围的字符。因此，如何打印 Unicode 字符呢？chr() 函数可以实现这种功能。例如，0x30A4 是日文中的一个片假名，它看起来就像是“ T ”，不过上面的横线是斜向上的：图 1. 日文片假名字符“ I ”（“ ee ”） 要打印这个片假名，后面再跟上一个笑脸符号，可以输入：perl -ebinmode(STDOUT, ":utf8"). print chr(0x30A4), chr(0x263a) 注意 binmode() 调用。如

果没有这个调用，或者等效的 -C 开关，Perl 就会打印一个打印语句中有宽数据的警告信息（详细信息请参阅 perldoc perlunicode 和 perldoc perlrun）。从我的观点来看，这只是 1% 的 Perl 用户需要使用的一个特性；其他 99% 的用户都需要忍受这个警告，或者额外编写一些代码来处理它。默认启用这个警告信息并不是什么好主意。 显然，如果您有很多东西需要输入，通过编码（或名字，Perl 也可以使用名字实现相同的功能）来指定字符实在是件痛苦的事情。如何才能以 UTF-8 的形式给 Perl 提供数据呢？script 2.pl 在脚本中就直接嵌入了 UTF-8 编码的数据。这个脚本的工作与 HERE 文档类似。Script 3.pl 会分别以英语、保加利亚语和俄语来打印“ 您好 ”。 不幸的是，UTF-8 数据并不被支持作为 HERE 标记（marker）。因此，script 4.pl 会报告一个错误“ Cant find string terminator \343 anywhere before EOF at ./4.pl line 7. ”（其中 \343 是一个无关字符）。 这可太糟了；单个 Unicode 字符可以作为 HERE 标记。它是简短的、富有表现力的且不太可能出现在文本中的字符（如果仔细选择的话。例如笑脸符号）。不幸的是，use utf8. 并不能使用 Unicode HERE 标记。您可以给一个变量分配一个 HERE 文档，而不是简单地打印这个变量，这样就需要在脚本中使用 use utf8.。识别 Unicode 字符 与打印 Unicode 数据相对的就是如何识别这些数据。内嵌的 ord() 函数如何执行？我在 脚本 5.pl 中对此逐一进行了检查，它可以很好地工作。从编码到字符的映射是一对一的，从字符到编码的映射也是一对一的。这是一个非常简单但却非常重要的测试；如何才能确保一个字符只会映射为一个惟一的编码呢？ 在命令行中使用任何编码（例如十进制的 500 或

1000）来运行 5.pl，结果如 5_out.txt 所示。 如果您所使用的编码与我在 5_out.txt 中使用的相同，就会注意到， 当编码太大时，这个脚本就什么也没有做；这正是我们期望的情况。在 5_out.txt 中，我使用单词 [GARBAGE] 来屏蔽那些无用的输出信息，但是您可以随意运行这个脚本，并查看输出结果。那么，我们应该如何识别命令行中输入的 Unicode 数据呢？脚本 6.pl 可以识别 @ARGV 中的 UTF-8 数据（6_out.txt 显示了一个运行 6.pl 的样例输出）。注意 -CA 开关（来自 perldoc perlrun），它告诉 Perl，@ARGV 中会包含 UTF-8 数据。如果没有 -CA 开关，Perl 就会认为输入数据是字符 199。还有，我执行了一个 join() 操作，然后又执行了一个 split()，将所有的 @ARGV 输入都变成字符数据。因此，对于 Unicode 字符的识别就可以很好地处理内嵌的数据、生成的数据以及命令行数据了。匹配和存储 Unicode 数据 Unicode 数据可以使用正则表达式进行匹配。在 split() 调用中我们就已经使用了这种技术，它可以匹配空字符串（它可以匹配任何地方，因此所匹配的字符串中的所有字符都会一个一个地返回）。那么如何才能匹配单个字符呢？脚本 Script 7.pl 就可以匹配正则表达式中的 Unicode 字符（7_out.txt 给出了运行 7.pl 的样例输出）。 这个脚本相当简单，但是它却展示了正则表达式中的 . (period) 字符可以真正匹配 Unicode 字符。您可以在 foreach() 调用之前添加 use bytes. 来查看一些错误的行为。这会告诉脚本按照字节进行匹配，这样您就会看到 3 行输出，而不是 7_out.txt 中那样的两行输出。为什么是 3 行呢？记住 UTF-8 编码的数据是可变长的格式；450 处的 Unicode 字符在使用 UTF-8 进行编码时实际上只需要两个字节。 下一个样例

脚本 8.pl 假定变量有一个 Unicode 名称，但是即使像 perldoc utf8 文档所建议的那样指定了 use utf8，也会得到 Unrecognized character 错误。因此，就假装 $data 是笑脸符号好了。Unicode 文本来自于 unicode-sample.txt 文件，也包含在下载文件中。 脚本 8.pl 可以正确匹配输入文本中的所有单词。如果您注释掉第一个和第二个 foreach() 代码行，就会看到 Perl 对输入数据进行了排序。脚本 9.pl 会将 Unicode 数据作为 hash 键来存储和读取。其工作原理就类似于 8.pl 使用 EOHIPPUS。脚本 9.pl 在以前的 Perl 5.6 中并不能正常工作，因此我很高兴它现在可以正常工作了。脚本 10.pl 也与 8.pl 一样， 不过它是将 Unicode 数据转换成大写和小写的形式。脚本 10.pl 可以完美地工作；所有的字母和重音符号都正确地转换了。结束语 Unicode 的工作非常有趣，也非常有必要。尽管目前对 Unicode 的支持还不是太好，但是在接下来的 5 年中，程序员的工具和编程体验中对 Unicode 的支持就会成为不可或缺的。尽管整个世界的联系越来越紧密，但是每个国家都不愿放弃自己的语言，因此程序员的任务就是不断适应这些变化。这不是说您应该学习新的语言，尽管这相当有趣，也非常值得。简单的预期是，应用程序可能需要进行本地化，并避免会将您限制到某一种语言的设计选择。例如，我们不能假定所有的文本都是从左到右排列的，也不能假定所有字符都是等宽的。 Perl 传统上就一直是一种面向文本的语言，因此 Unicode 支持改变了该语言的关键特性。正如本文介绍的一样，5.8.6 版本的 Perl 可以很好地处理 Unicode。Perl 6 会为了处理 Unicode 而从头进行设计，因此我们可以期望能够获得 2 级或 3 级的技术支持，而不像在 Perl 5 中只提供了部

分的 1 级支持。 对 Unicode 支持的代价是处理数据时需要花费更长的时间。程序不再假设一个字节就是一个字符，因此所有的数据都需要从 UTF-8 进行解码，然后再重新编码成 UTF-8。（注意 Perl 5 可以支持其他编码；请参阅 perldoc encoding，但是 UTF-8 是 Unicode 中最流行的一种编码，我也只推荐使用这种编码）。 我建议您开始面对这种挑战，并且编写能够处理 Unicode 的程序。与量子计算和遗传算法一样（我在 IBM 的 developerWorks 专栏中也介绍了这两个主题的内容），我认为对于将来的计算挑战来说，Unicode 是目前最重要的领域之一。